EXHIBIT 2

EXHIBIT C
Supplemental Infringement Contentions for the ’702 Patent

NOTE: The infringement evidence cited below is exemplary and not exhaustive. The cited examples are taken from Android 2.2, 2.3, and Google’s Android websites. Oracle’s infringement contentions apply to all versions of Android having similar or nearly identical code or documentation, including past and expected future releases. Although Oracle’s investigation is ongoing, the ’702 patent is infringed by all versions of Android from Oct. 21, 2008 to the present, including Android 1.1, 1.5 (“Cupcake”), 1.6 (“Donut”), 2.0/2.1 (“Éclair”), 2.2 (“Froyo”), and 2.3 (“Gingerbread”).

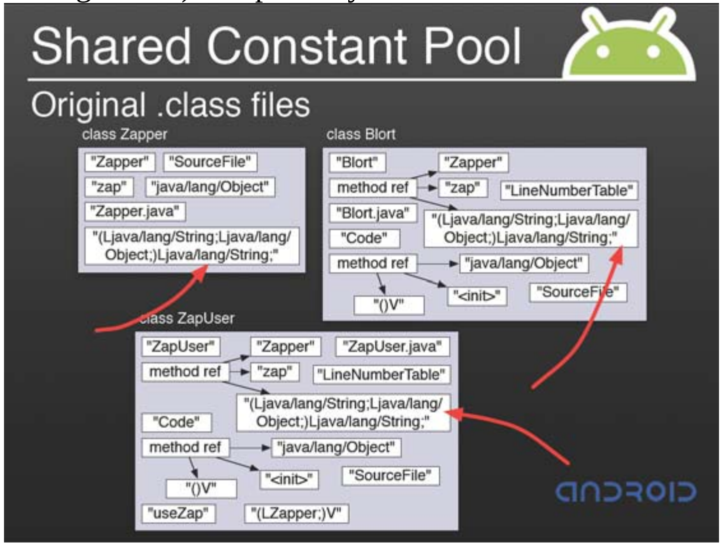
The cited source code examples are taken from http://android.git.kernel.org/. The citations are shortened and mirror the file paths shown in http://android.git.kernel.org/. For example, “dalvik\vm\native\InternalNative.c” maps to “[platform/dalvik.git] / vm / native / InternalNative.c” (accessible at http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/native/InternalNative.c). Google has apparently made modifications to certain source code files and directories since Oracle’s Preliminary Infringement Contentions were served on December 2, 2010. As such, file paths may in some cases refer to earlier versions of Android than what is immediately available at http://android.git.kernel.org/.

It appears that the Android git source code repository (accessible through http://android.git.kernel.org/) was created on or around Oct. 21, 2008. As such, the list of infringing Android versions may be expanded based on what Oracle learns about earlier Android versions.

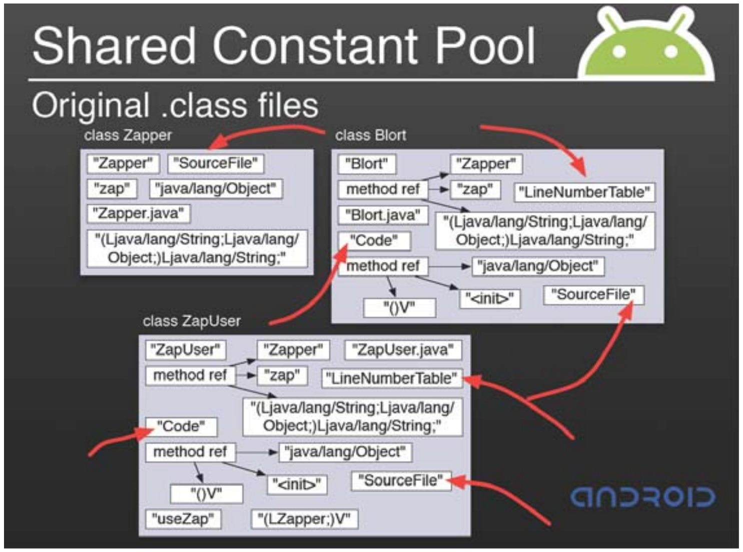
The asserted claims include apparatus, method, and computer-readable medium claims. Anyone who makes, uses, offers to sell, sells, or imports the computers running the Android SDK within or into the United States directly infringes the apparatus claims. Similarly, anyone who engages in the above conduct with respect to storage devices containing the Android SDK directly infringes the computer-readable medium claims. Anyone who uses the Android SDK directly infringes the method claims. Thus Google and its downstream licensees, including device manufacturers and application developers, directly infringe. Google induces and contributes to infringement of all asserted claims by distributing the Android SDK with the intention that it will be executed by developers. The Android code cited below necessarily infringes because developers must run the Android dx tool to build Android applications, and generate Android bytecode and .dex files, and run the Dalvik virtual machine to test them. The Android SDK is a tool used purely to build and test Android programs. It is neither a staple article nor capable of substantial non-infringing use. Google supplies the Android SDK in and from the United States.

When infringement evidence first presented with respect to one claim is referred to with respect to another, the evidence is applicable because it is not limited to a particular form of infringement.

The ’702 Patent	Infringed By
1. A method of pre-processing class files comprising:	The Android dx tool involves a method of pre-processing .class files into a Dalvik executable format (.dex) file. **“dx** The dx tool lets you generate Android bytecode from .class files. The tool converts target files and/or directories to Dalvik executable format (.dex) files, so that they can run in the Android environment.” Android Developer Tools available at http://developer.android.com/guide/developing/tools/othertools.html The method of pre-processing class files into a .dex file that can be interpreted by the Dalvik Virtual Machine (Dalvik VM) is explained in the Dalvik VM video presentation and related presentation from Google I/O 2008, dated 5/29/2008. *See* Google I/O 2008 Video entitled “*Google I/O 2008 - Dalvik Virtual Machine Internals*,” presented by Dan Bornstein, http://developer.android.com/videos/index.html#v=ptjedOZEXPM (“Dalvik Video”), at time 5:45–10:45. *See also* Google I/O 2008 Presentation Slides, entitled, “*Dalvik Virtual Machine Internals, Google I/O 2008*,” presented by Dan Bornstein (“Dalvik Presentation”) at slides 11-22, available at http://sites.google.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf?attredirects=0. In the Android source code, *see generally*: “Classes for translating Java classfiles into Dalvik classes. PACKAGES USED: • com.android.dx.cf.code

The ’702 Patent	Infringed By
	• com.android.dx.cf.direct • com.android.dx.cf.iface • com.android.dx.dex.code • com.android.dx.dex.file • com.android.dx.rop.code • com.android.dx.rop.cst • com.android.dx.util” dalvik\dx\src\com\android\dx\dex\cf\package.html.
determining plurality of duplicated elements in a plurality of class files;	The Android dx tool determines a plurality of duplicated elements in a plurality of class files, as explained in the Dalvik Video at time 7:50-8:45 and Dalvik Presentation, slides 18-19. The Dalvik Presentation shows the determination of a plurality of duplicated elements (e.g., class signatures and string names) in a plurality of class files:  Shared Constant Pool Original .class files class Zapper "Zapper" "SourceFile" "zap" "java/lang/Object" "Zapper.java" "(Ljava/lang/String;Ljava/lang/Object;)Ljava/lang/String;" class Blort "Blort" "Zapper" method ref "zap" "LineNumberTable" "Blort.java" "(Ljava/lang/String;Ljava/lang/Object;)Ljava/lang/String;" "Code" method ref "java/lang/Object" "<init>" "SourceFile" "()V" class ZapUser "ZapUser" "Zapper" "ZapUser.java" method ref "zap" "LineNumberTable" "(Ljava/lang/String;Ljava/lang/Object;)Ljava/lang/String;" "Code" method ref "java/lang/Object" "<init>" "SourceFile" "()V" "useZap" "(LZapper;)V" android (Dalvik Presentation, slide 18) (Shows identification of common class signatures in the class files)

<table>
<tr><th>The ’702 Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>



Shared Constant Pool

Original .class files

class Zapper

"Zapper" "SourceFile" "zap" "java/lang/Object" "Zapper.java" "(Ljava/lang/String;Ljava/lang/Object;)Ljava/lang/String;"

class Blort

"Blort" "Zapper" method ref "zap" "LineNumberTable" "Blort.java" "(Ljava/lang/String;Ljava/lang/Object;)Ljava/lang/String;" "Code" method ref "java/lang/Object" "()V" "<init>" "SourceFile"

class ZapUser

"ZapUser" "Zapper" "ZapUser.java" method ref "zap" "LineNumberTable" "(Ljava/lang/String;Ljava/lang/Object;)Ljava/lang/String;" "Code" method ref "java/lang/Object" "()V" "<init>" "SourceFile" "useZap" "(LZapper;)V"

(Dalvik Presentation, slide 19)
(Shows identification of common string names in the class files)

In the Android source code, *see also generally*:

> “Interfaces and implementation of things related to the constant pool.
> PACKAGES USED:
>   * com.android.dx.rop.type
>   * com.android.dx.util”

dalvik/dx/src/com/android/dx/rop/cst/package.html.

See also DexFile.java:

```

    /**
     * Gets the {@link IndexedItem} corresponding to the given constant,
     * if it is a constant that has such a correspondence, or return
```

</td></tr>
</table>

The ’702 Patent	Infringed By

```
     * {@code null} if it isn't such a constant. This will throw
     * an exception if the given constant <i>should</i> have been found
     * but wasn't.
     *
     * @param cst {@code non-null;} the constant to look up
     * @return {@code null-ok;} its corresponding item, if it has a corresponding
     * item, or {@code null} if it's not that sort of constant
     */
    /*package*/ IndexedItem findItemOrNull(Constant cst) {
        IndexedItem item;

        if (cst instanceof CstString) {
            return stringIds.get(cst);
        } else if (cst instanceof CstType) {
            return typeIds.get(cst);
        } else if (cst instanceof CstBaseMethodRef) {
            return methodIds.get(cst);
        } else if (cst instanceof CstFieldRef) {
            return fieldIds.get(cst);
        } else {
            return null;
        }
    }

    /**
     * Returns the contents of this instance as a {@code .dex} file,
     * in a {@link ByteArrayAnnotatedOutput} instance.
     *
     * @param annotate whether or not to keep annotations
     * @param verbose if annotating, whether to be verbose
     * @return {@code non-null;} a {@code .dex} file for this instance
     */
    private ByteArrayAnnotatedOutput toDex0(boolean annotate,
            boolean verbose) {
        /*
         * The following is ordered so that the prepare() calls which
         * add items happen before the calls to the sections that get
         * added to.
         */

        classDefs.prepare();
        classData.prepare();
        wordData.prepare();
```

The ’702 Patent	Infringed By

```
        byteData.prepare();
        methodIds.prepare();
        fieldIds.prepare();
        protoIds.prepare();
        typeLists.prepare();
        typeIds.prepare();
        stringIds.prepare();
        stringData.prepare();
        header.prepare();

        // Place the sections within the file.

        int count = sections.length;
        int offset = 0;

        for (int i = 0; i < count; i++) {
            Section one = sections[i];
            int placedAt = one.setFileOffset(offset);
            if (placedAt < offset) {
                throw new RuntimeException("bogus placement for section " + i);
            }

            try {
                if (one == map) {
                    /*
                     * Inform the map of all the sections, and add it
                     * to the file. This can only be done after all
                     * the other items have been sorted and placed.
                     */
                    MapItem.addMap(sections, map);
                    map.prepare();
                }

                if (one instanceof MixedItemSection) {
                    /*
                     * Place the items of a MixedItemSection that just
                     * got placed.
                     */
                    ((MixedItemSection) one).placeItems();
                }

                offset = placedAt + one.writeSize();
            } catch (RuntimeException ex) {
```

The ’702 Patent	Infringed By

```
                throw ExceptionWithContext.withContext(ex,
                        "...while writing section " + i);
            }
        }

        // Write out all the sections.

        fileSize = offset;
        byte[] barr = new byte[fileSize];
        ByteArrayAnnotatedOutput out = new ByteArrayAnnotatedOutput(barr);

        if (annotate) {
            out.enableAnnotations(dumpWidth, verbose);
        }

        for (int i = 0; i < count; i++) {
            try {
                Section one = sections[i];
                int zeroCount = one.getFileOffset() - out.getCursor();
                if (zeroCount < 0) {
                    throw new ExceptionWithContext("excess write of " +
                            (-zeroCount));
                }
                out.writeZeroes(one.getFileOffset() - out.getCursor());
                one.writeTo(out);
            } catch (RuntimeException ex) {
                ExceptionWithContext ec;
                if (ex instanceof ExceptionWithContext) {
                    ec = (ExceptionWithContext) ex;
                } else {
                    ec = new ExceptionWithContext(ex);
                }
                ec.addContext("...while writing section " + i);
                throw ec;
            }
        }

        if (out.getCursor() != fileSize) {
            throw new RuntimeException("foreshortened write");
        }

        // Perform final bookkeeping.

        calcSignature(barr);
```

The ’702 Patent	Infringed By

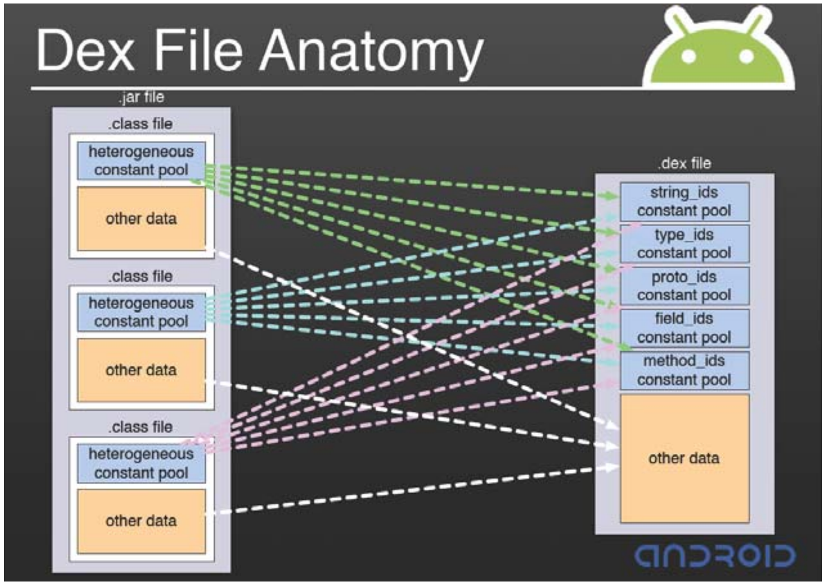
```
        calcChecksum(barr);

        if (annotate) {
            wordData.writeIndexAnnotation(out, ItemType.TYPE_CODE_ITEM,
                    "\nmethod code index:\n\n");
            getStatistics().writeAnnotation(out);
            out.finishAnnotating();
        }

        return out;
    }

    /**
     * Generates and returns statistics for all the items in the file.
     *
     * @return {@code non-null;} the statistics
     */
    public Statistics getStatistics() {
        Statistics stats = new Statistics();

        for (Section s : sections) {
            stats.addAll(s);
        }

        return stats;
    }

    /**
     * Calculates the signature for the {@code .dex} file in the
     * given array, and modify the array to contain it.
     *
     * @param bytes {@code non-null;} the bytes of the file
     */
    private static void calcSignature(byte[] bytes) {
        MessageDigest md;

        try {
            md = MessageDigest.getInstance("SHA-1");
        } catch (NoSuchAlgorithmException ex) {
            throw new RuntimeException(ex);
        }

        md.update(bytes, 32, bytes.length - 32);

```

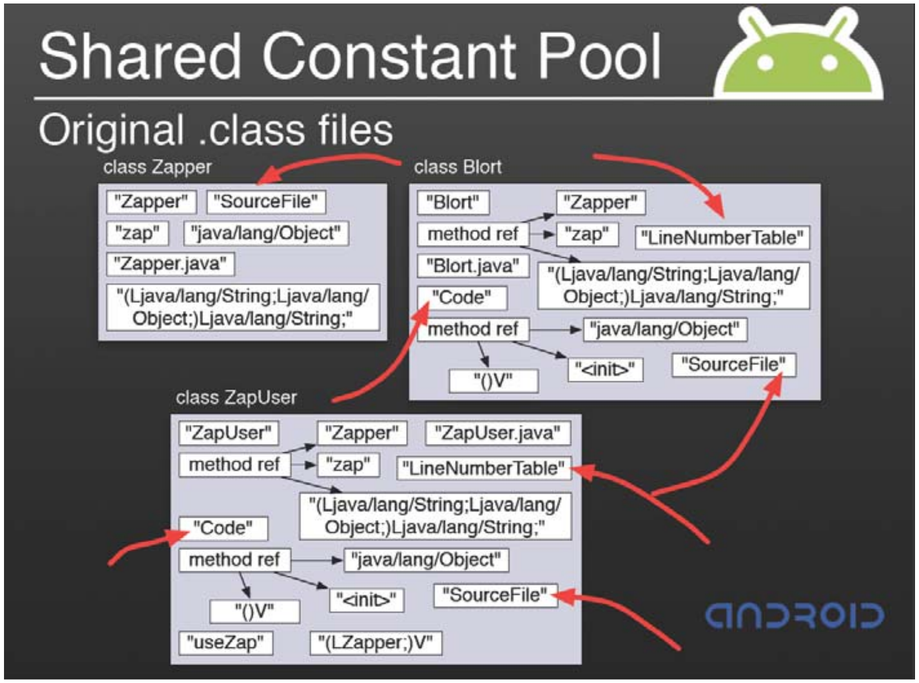
<table>
<tr><th>The ’702 Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>

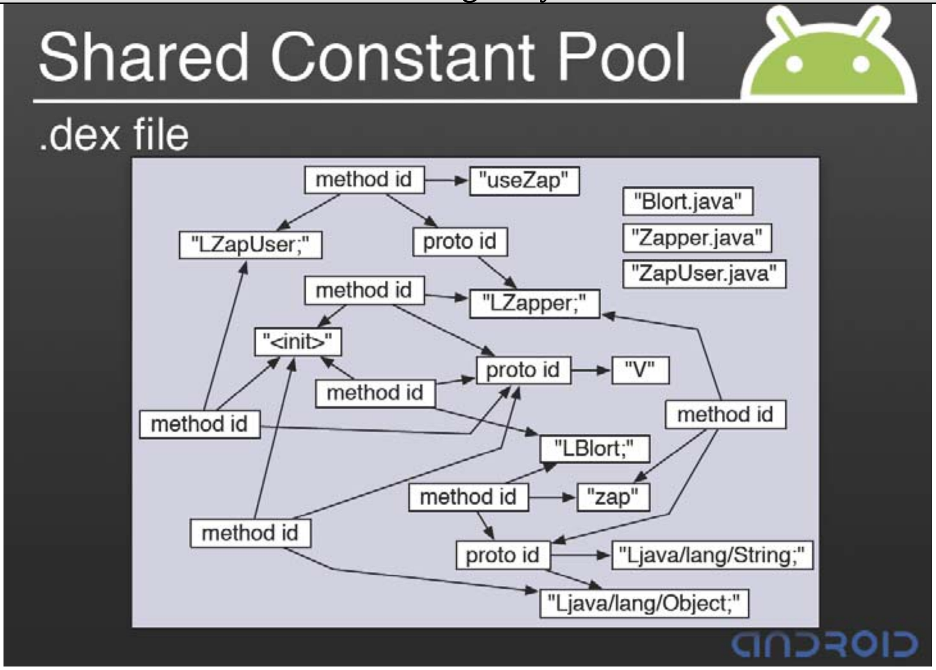
```
        try {
            int amt = md.digest(bytes, 12, 20);
            if (amt != 20) {
                throw new RuntimeException("unexpected digest write: " + amt +
                                           " bytes");
            }
        } catch (DigestException ex) {
            throw new RuntimeException(ex);
        }
    }

    /**
     * Calculates the checksum for the {@code .dex} file in the
     * given array, and modify the array to contain it.
     *
     * @param bytes {@code non-null;} the bytes of the file
     */
    private static void calcChecksum(byte[] bytes) {
        Adler32 a32 = new Adler32();

        a32.update(bytes, 12, bytes.length - 12);

        int sum = (int) a32.getValue();

        bytes[8]  = (byte) sum;
        bytes[9]  = (byte) (sum >> 8);
        bytes[10] = (byte) (sum >> 16);
        bytes[11] = (byte) (sum >> 24);
    }
}
```

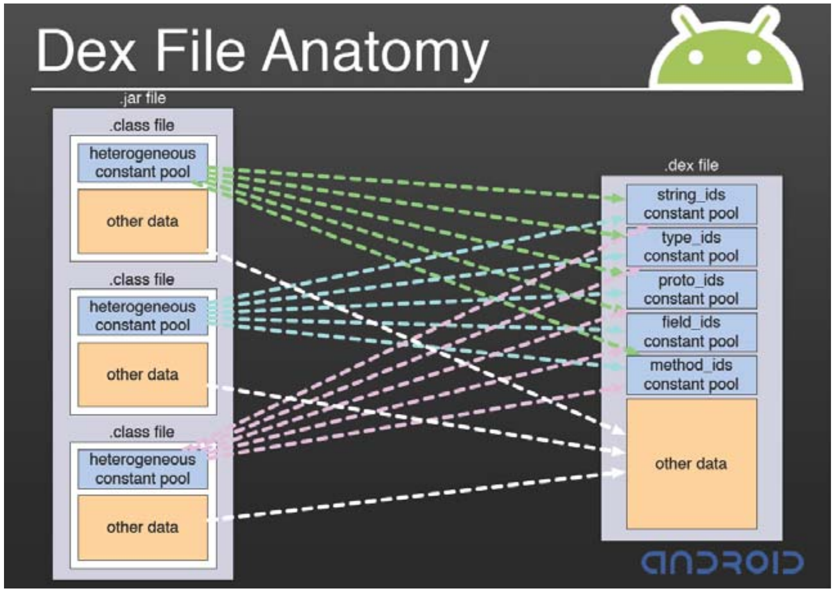
dalvik/dx/src/com/android/dx/dex/file/DexFile.java.

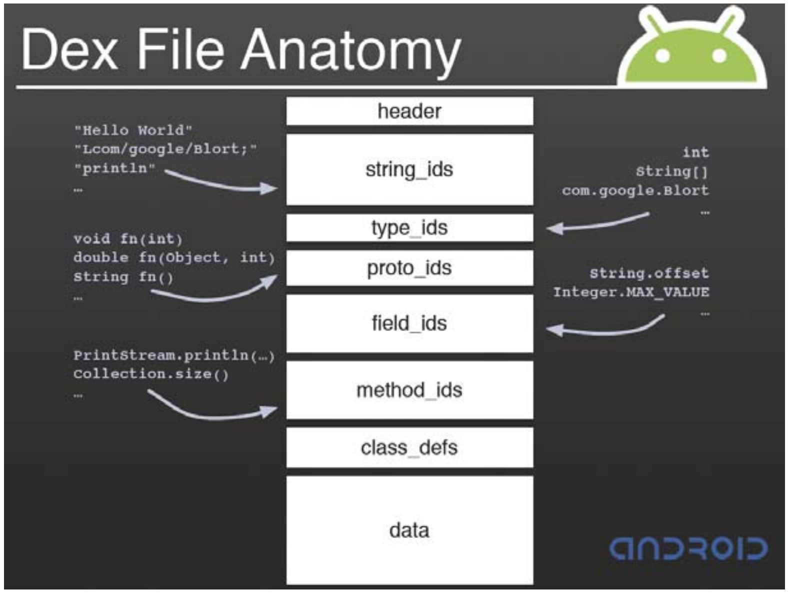
See also:
dalvik/dx/src/com/android/dx/dex/file/TypeIdsSection.java
dalvik/dx/src/com/android/dx/dex/file/TypeIdItem.java
dalvik/dx/src/com/android/dx/cf/cst/ConstantPoolParser.java</td></tr>
<tr><td>forming a shared table comprising said plurality of duplicated</td><td>The Android dx tool forms a shared table of the duplicated elements from the plurality of class files. This process is explained in the Dalvik Video at time 7:20–9:25 and Dalvik</td></tr>
</table>

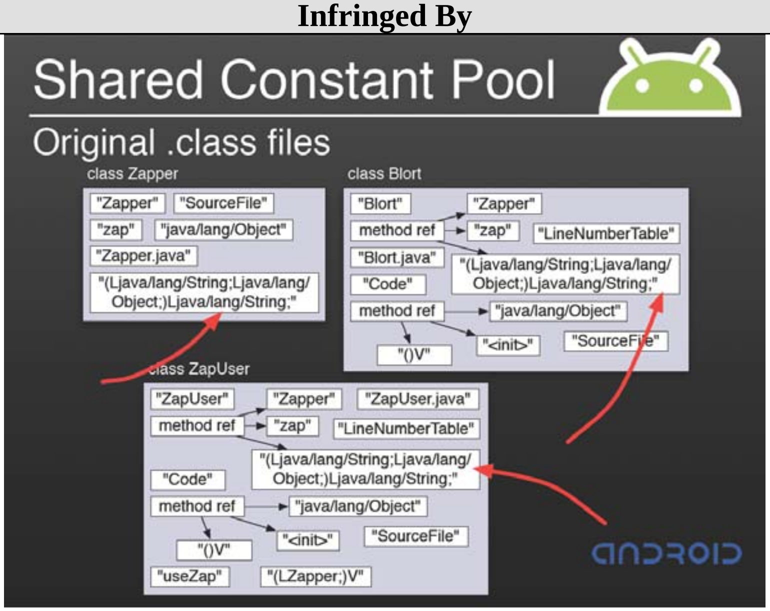
The ’702 Patent	Infringed By
elements;	Presentation, slides 15-20, where the recited shared table includes, e.g., one or more of the “string_ids constant pool,” “type_ids constant pool,” “proto_ids constant pool,” “field_ids constant pool,” and “method_ids constant pool.” The Dalvik Presentation shows the elements of the class files combining into a shared constant pool (shared tables) in the .dex file.  Dex File Anatomy .jar file .class file heterogeneous constant pool other data .class file heterogeneous constant pool other data .class file heterogeneous constant pool other data .dex file string_ids constant pool type_ids constant pool proto_ids constant pool field_ids constant pool method_ids constant pool other data android (Dalvik Presentation, slide 15) In the illustration above, each of “string_ids,” “type_ids” and “method_ids” are examples of the shared tables (or, equivalently, a collective shared table). In addition, the discussion of the “Shared Constant Pool” in the Dalvik Video explains that the duplicated elements in the class files are consolidated into the shared constant pool (shared table) of the .dex file. *See* Dalvik Presentation, slides 15-21. For example, slide 19 of the Dalvik Presentation shows the separate class files having

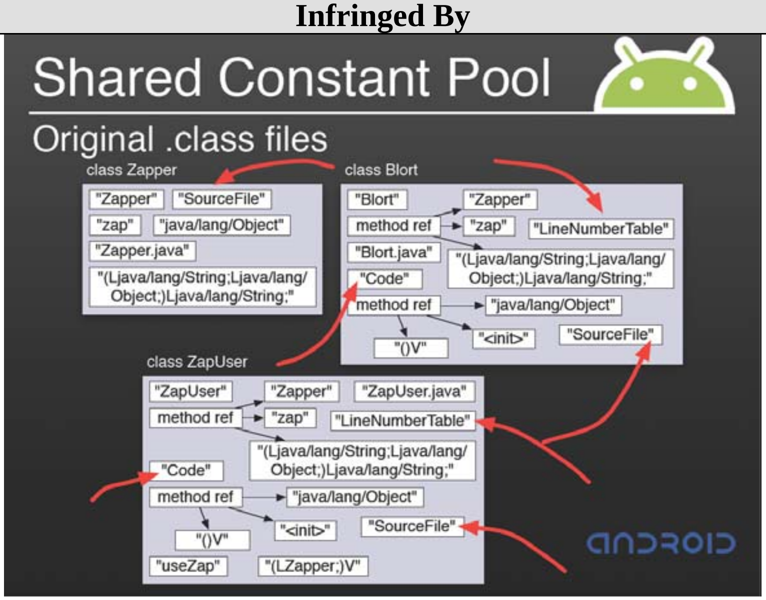
The ’702 Patent	Infringed By
	duplicated elements.  Shared Constant Pool Original .class files class Zapper "Zapper" "SourceFile" "zap" "java/lang/Object" "Zapper.java" "(Ljava/lang/String;Ljava/lang/Object;)Ljava/lang/String;" class Blort "Blort" "Zapper" method ref "zap" "LineNumberTable" "Blort.java" "(Ljava/lang/String;Ljava/lang/Object;)Ljava/lang/String;" "Code" method ref "java/lang/Object" "<init>" "SourceFile" "()V" class ZapUser "ZapUser" "Zapper" "ZapUser.java" method ref "zap" "LineNumberTable" "(Ljava/lang/String;Ljava/lang/Object;)Ljava/lang/String;" "Code" method ref "java/lang/Object" "<init>" "SourceFile" "()V" "useZap" "(LZapper;)V" (Dalvik Presentation, slide 19) Next, slide 20 of the Dalvik Presentation shows a representation of the class files after being processed into a single .dex file, with the duplicate elements removed; the elements are then stored in a shared constant pool (shared table):

The ’702 Patent	Infringed By
	 Shared Constant Pool .dex file method id "useZap" "Blort.java" "LZapUser;" proto id "Zapper.java" "ZapUser.java" method id "LZapper;" "<init>" proto id "V" method id method id method id "LBlort;" method id "zap" method id proto id "Ljava/lang/String;" "Ljava/lang/Object;" android (Dalvik Presentation, slide 20) In the Android source code, *see also generally*: “Interfaces and implementation of things related to the constant pool. PACKAGES USED: * com.android.dx.rop.type * com.android.dx.util” dalvik/dx/src/com/android/dx/rop/cst/package.html. *See also:* dalvik/dx/src/com/android/dx/dex/file/DexFile.java dalvik/dx/src/com/android/dx/dex/file/TypeIdsSection.java

The ’702 Patent	Infringed By
	dalvik/dx/src/com/android/dx/dex/file/TypeIdItem.java dalvik/dx/src/com/android/dx/cf/cst/ConstantPoolParser.java
removing said duplicated elements from said plurality of class files to obtain a plurality of reduced class files; and	The Android dx tool removes the duplicated elements from the plurality of class files (e.g., as part of the process of forming the .dex file) and obtains a plurality of reduced class files (the reduced class files including a subset of the code and data contained in the class files). This process, and contents of the reduced class file, is clearly explained and illustrated in the Dalvik Video at time 7:20–9:25 and Dalvik Presentation, slides 15-20. The Dalvik Presentation shows the class files combining into a shared constant pool (shared table) in the .dex file, whereby duplicated elements are removed from the class files when using a subset of the code and data contained in the class files, i.e., the reduced class files, to form the .dex file.  Dex File Anatomy .jar file .class file heterogeneous constant pool other data .class file heterogeneous constant pool other data .class file heterogeneous constant pool other data .dex file string_ids constant pool type_ids constant pool proto_ids constant pool field_ids constant pool method_ids constant pool other data (Dalvik Presentation, slide 15)

The ’702 Patent	Infringed By
	The original class files are combined into a single .dex file, which includes a plurality of reduced class files (i.e., a subset of code and data of the class files, with duplicates removed). This is also illustrated in slide 11 of the Dalvik presentation, which shows the anatomy of a .dex file:  Dex File Anatomy header string_ids type_ids proto_ids field_ids method_ids class_defs data "Hello World" "Lcom/google/Blort;" "println" … void fn(int) double fn(Object, int) String fn() … PrintStream.println(…) Collection.size() … int String[] com.google.Blort … String.offset Integer.MAX_VALUE … android (Dalvik Presentation, slide 11) Next, slides 18-20 of the Dalvik Presentation show the removal of the duplicated elements of the plurality of class files such that the resulting .dex file contains only one copy of each element in its shared constant pool (shared table).

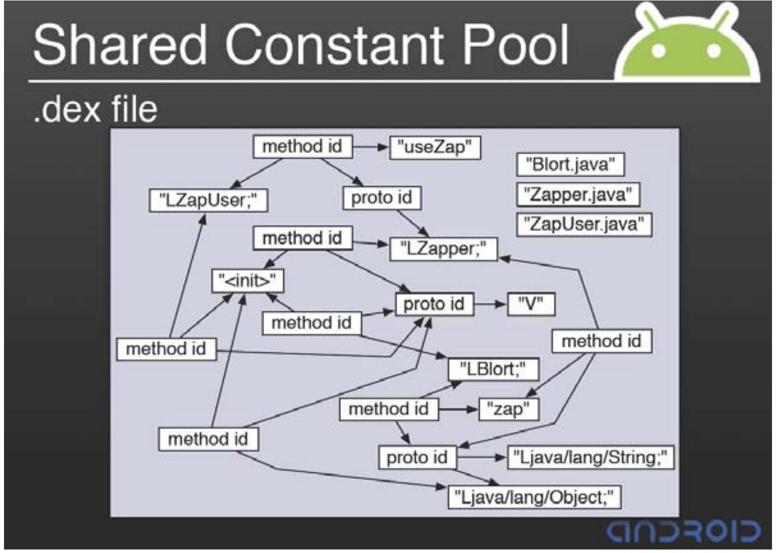
The ’702 Patent	Infringed By
	 Shared Constant Pool Original .class files class Zapper "Zapper" "SourceFile" "zap" "java/lang/Object" "Zapper.java" "(Ljava/lang/String;Ljava/lang/Object;)Ljava/lang/String;" class Blort "Blort" "Zapper" method ref "zap" "LineNumberTable" "Blort.java" "(Ljava/lang/String;Ljava/lang/Object;)Ljava/lang/String;" "Code" method ref "java/lang/Object" "<init>" "SourceFile" "()V" class ZapUser "ZapUser" "Zapper" "ZapUser.java" method ref "zap" "LineNumberTable" "(Ljava/lang/String;Ljava/lang/Object;)Ljava/lang/String;" "Code" method ref "java/lang/Object" "<init>" "SourceFile" "()V" "useZap" "(LZapper;)V" (Dalvik Presentation, slide 18)

<table>
<tr><th>The ’702 Patent</th><th>Infringed By</th></tr>
<tr><td></td><td>


Shared Constant Pool

Original .class files

(Dalvik Presentation, slide 19)

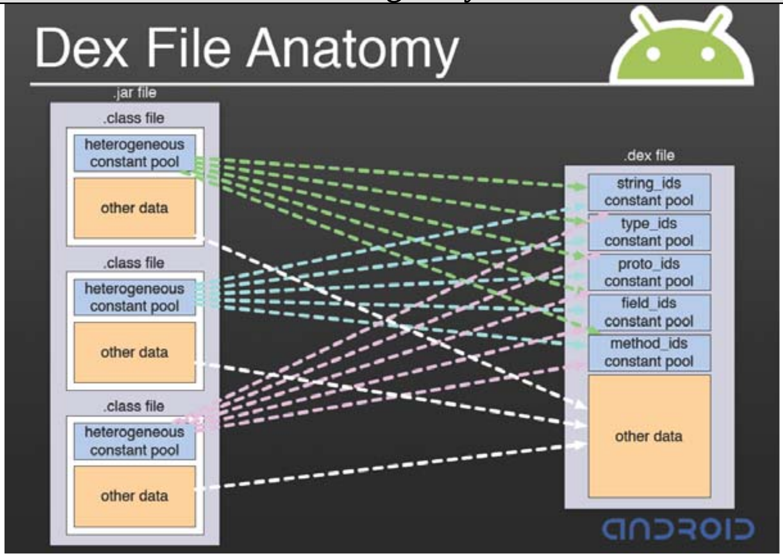
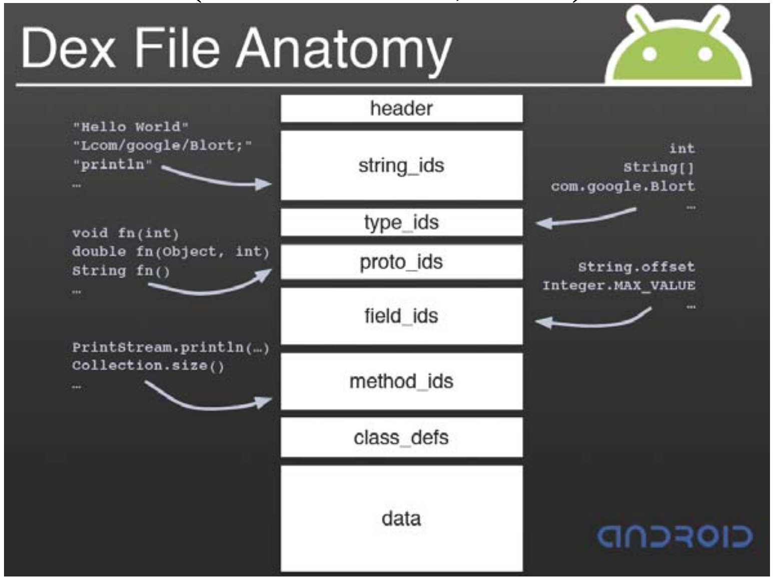


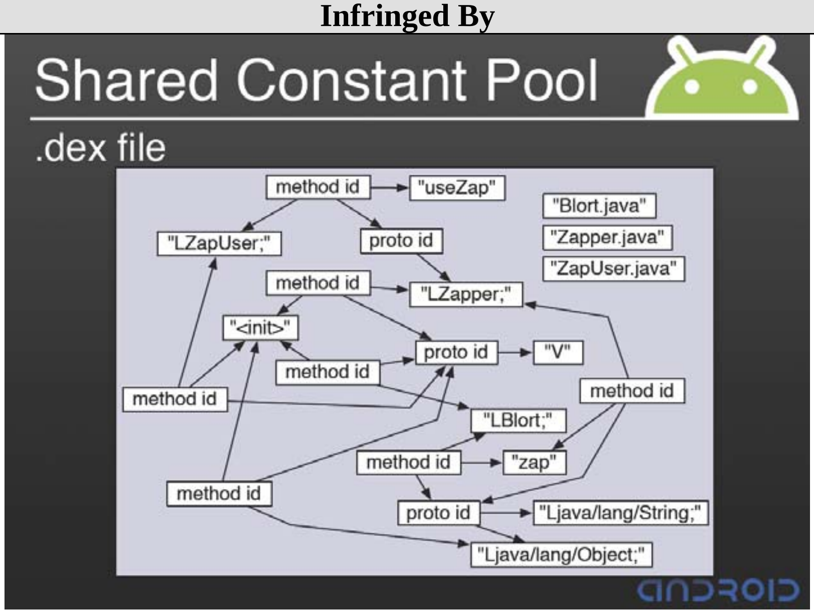
Shared Constant Pool

.dex file

(Dalvik Presentation, slide 20)
</td></tr>
</table>

The ’702 Patent	Infringed By
	In the Android source code, *see also generally*: “Interfaces and implementation of things related to the constant pool. PACKAGES USED: * com.android.dx.rop.type * com.android.dx.util” dalvik/dx/src/com/android/dx/rop/cst/package.html. *See also:* dalvik/dx/src/com/android/dx/dex/file/DexFile.java dalvik/dx/src/com/android/dx/dex/file/TypeIdsSection.java dalvik/dx/src/com/android/dx/dex/file/TypeIdItem.java dalvik/dx/src/com/android/dx/cf/cst/ConstantPoolParser.java
forming a multi-class file comprising said plurality of reduced class files and said shared table.	As explained above, the Android dx tool forms a multi-class file—the .dex file—comprising the reduced class files and a shared constant pool (shared table) such that duplicate elements have been removed. This process is explained in the Dalvik Video at time 7:20–9:25 and Dalvik Presentation, slides 11 and 15-20. The reduced class files include a subset of the code and data of the original class files, e.g., “class_defs” and “data” illustrated in slide 11 and the “other data” illustrated in slide 15, and the recited shared table includes, e.g., one or more of the “string_ids constant pool,” “type_ids constant pool,” “proto_ids constant pool,” “field_ids constant pool,” and “method_ids constant pool.” The Dalvik Presentation shows the original class files being combined into a .dex file (multi-class file) comprising the plurality of reduced class files and the shared constant pool (shared table):

The ’702 Patent	Infringed By
	 Dex File Anatomy .jar file .class file heterogeneous constant pool other data .class file heterogeneous constant pool other data .class file heterogeneous constant pool other data .dex file string_ids constant pool type_ids constant pool proto_ids constant pool field_ids constant pool method_ids constant pool other data android (Dalvik Presentation, slide 15)  Dex File Anatomy "Hello World" "Lcom/google/Blort;" "println" … void fn(int) double fn(Object, int) String fn() … PrintStream.println(…) Collection.size() … header string_ids type_ids proto_ids field_ids method_ids class_defs data int String[] com.google.Blort … String.offset Integer.MAX_VALUE … android (Dalvik Presentation, slide 11)

The ’702 Patent	Infringed By
	 Shared Constant Pool .dex file method id "useZap" "Blort.java" "Zapper.java" "ZapUser.java" "LZapUser;" proto id method id "LZapper;" "<init>" proto id "V" method id method id method id "LBlort;" method id "zap" method id proto id "Ljava/lang/String;" "Ljava/lang/Object;" android (Dalvik Presentation, slide 20) In the Android source code, *see generally*: “Classes for translating Java classfiles into Dalvik classes. PACKAGES USED: • com.android.dx.cf.code • com.android.dx.cf.direct • com.android.dx.cf.iface • com.android.dx.dex.code • com.android.dx.dex.file • com.android.dx.rop.code • com.android.dx.rop.cst • com.android.dx.util” dalvik\dx\src\com\android\dx\dex\cf\package.html.

The ’702 Patent	Infringed By
	See also: /** * Representation of an entire {@code .dex} (Dalvik EXecutable) * file, which itself consists of a set of Dalvik classes. */ public final class DexFile { /** {@code non-null;} word data section */ private final MixedItemSection wordData; dalvik\dx\src\com\android\dx\dex\file\DexFile.java. *See also:* dalvik/dx/src/com/android/dx/dex/file/DexFile.java dalvik/dx/src/com/android/dx/dex/file/TypeIdsSection.java dalvik/dx/src/com/android/dx/dex/file/TypeIdItem.java dalvik/dx/src/com/android/dx/cf/cst/ConstantPoolParser.java

The ’702 Patent	Infringed By
5. The method of claim 1, wherein said step of determining a plurality of duplicated elements comprises:	*See* Claim 1, *supra*.
determining one or more constants shared between two or more class files.	The Android dx tool determines constants shared between two or more class files. This process is explained in the Dalvik Video at time 7:20-9:25 and Dalvik Presentation, slides 11-20. The Dalvik Presentation shows the elements of the class files identified for combining into a shared constant pool (shared tables) in the .dex file.